"Express Mail" mailing label number EV 164034058 US

Date of Deposit: November 12, 2003

Attorney Docket No.14841US02

LOW MEMORY AND MIPS EFFICIENT TECHNIQUE FOR DECODING HUFFMAN CODES USING MULTI-STAGE, MULTI-BITS LOOKUP AT DIFFERENT LEVELS

RELATED APPLICATIONS

[001] This application claims priority to U.S. Provisional Patent Application Serial No. _____, entitled "**LOW MEMORY AND MIPS EFFICIENT TECHNIQUE FOR DECODING HUFFMAN CODES USING MULTI-STAGE, MULTI-BITS LOOKUP AT DIFFERENT LEVELS**", filed November 7, 2003, by Singhal, which is incorporate herein by reference.

FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

[002] [Not Applicable]

[MICROFICHE/COPYRIGHT REFERENCE]

[003] [Not Applicable]

BACKGROUND OF THE INVENTION

[004] Huffman coding is a loss-less compression technique often used in lossy compression schemes as the final step after decomposition and quantization of a signal. Huffman coding uses unique variable length code words and no Huffman code is a prefix of another Huffman code. For a given probability density function of a symbol set, shorter codes can be assigned to frequently occurring symbols while longer codes can be assigned to less frequently occurring symbols. Huffman's minimum redundancy encoding minimizes

the average number of bits required to represent the data and is one type of variable length coding.

[005] Huffman coding is useful for reduction of the bit-rate by exploring statistical redundancies and to encode a "minimum set" of information using entropy-coding technique. Usually entropy coders exploit the symbol probabilities independent of the previous symbols, and hence are optimal for uncorrelated sequences.

[006] The Huffman coding/decoding is one of the key components of an encoder/decoder in many of the audio/video compression standards. So there is a necessity of its implementation on the digital signal processor (DSP) for a cost effective solution. However, the architecture and instruction set of DSPs are optimized for computations with operands that are byte, half word or word size. Since the symbols have variable length after encoding, there is a necessity of extracting code words that are not necessarily byte, half word, or word size.

[007] On the other hand, the memory access usually fetches data that are only byte, half word, or word aligned. Due to this, the speed of the Huffman Decoder implemented on DSP is lower than that of a corresponding implementation on dedicated hardware.

[008] In addition to this, most of the compression standards have multiple Huffman tables containing long code words that have to be stored on expensive on-chip memory for fast access. These two factors emphasize the importance of high speed and memory efficient implementation of Huffman decoding on the DSP. Though the DSP's architectures are optimized for for signal processing applications, they are not so in the case of search algorithms.

[009] Given that Huffman encoding/decoding is a very important component of compression standards, it is important that the Huffman Encoder/Decoder should be efficiently implemented on the DSP chosen. The complexity of the Huffman decoder's implementation lies in fast search of the symbol encoded from the bit-stream without consuming large memory. These two requirements are conflicting and in addition, the standards have multiple tables of large code length.

[0010] Further limitations and disadvantages of conventional and traditional systems will become apparent to one of skill in the art through comparison of such systems with the invention as set forth in the remainder of the present application with reference to the drawings.

BRIEF SUMMARY OF THE INVENTION

[0011] Presented herein is a low memory and MIPS efficient technique for decoding Huffman codes using multi-stage, multi-bits lookup at different levels.

[0012] In one embodiment, there is a method for storing a variable length code table in a memory. The method comprises calculating a proportion for each level of a binary tree, the proportion for each level being the quotient of a number of nodes at each level of the binary tree and the number of possible nodes at each level of the binary tree, comparing the proportion for each level of the binary tree to a threshold value which is fine tuned depending upon the distribution of various Huffman codes at various levels in the binary tree and generating at least one new binary tree from at least one node at a particular level, if the threshold exceeds the proportion for a next level.

[0013] In another embodiment, there is an article of manufacture comprising a computer readable medium. The computer readable medium stores a plurality of instructions. Execution of the plurality of instructions causes calculating a proportion for each level of a binary tree, the proportion for each level being the quotient of a number of nodes at each level of the binary tree and the number of possible nodes at each level of the binary tree, comparing the proportion for each level of the binary tree to a threshold value, and generating at least one new binary tree from at least one node at a particular level, if the threshold exceeds the proportion for a next level.

[0014] These and other advantages and novel features of the present invention, as well as details of an illustrated embodiment thereof, will be more fully understood from the following description and drawings.

BRIEF DESCRIPTION OF SEVERAL VIEWS OF THE DRAWINGS

[0015] **FIGURE 1** is a block diagram describing an exemplary Huffman code;

[0016] **FIGURE 2** is a block diagram of a binary tree for the Huffman code in FIGURE 1;

[0017] **FIGURE 3** is a block diagram describing the packing of the binary tree in accordance with an embodiment of the present invention;

[0018] **FIGURE 4** is a flow diagram for packing a binary tree for a Huffman code in accordance with an embodiment of the present invention;

[0019] **FIGURE 5** is a flow diagram for decoding a Huffman code in accordance with an embodiment of the present invention;

[0020] **FIGURE 6** is a block diagram of an exemplary hardware environment wherein the present invention can be practiced;

[0021] **FIGURE 7** is a block diagram of a decoder system in accordance with an embodiment of the present invention; and

[0022] **FIGURE 8** is a block diagram of an audio/video decoder in accordance with an embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

[0023] Referring now to **FIGURE 1**, there is illustrated a block diagram describing an exemplary Huffman code. Huffman coding is a loss-less compression technique that uses unique variable length code words to encode different symbols. For a given probability density function Huffman uses unique variable length code words, such that no code is a prefix for another code.

[0024] The Huffman code comprises 12 data symbols A,B,...L, along with corresponding Huffman codes. The 12 data symbols can be packed in various ways but each combination gives different memory requirements and requires a different number of lookups. In one way, the 12 data symbols can be packed in a total of 32 memory locations. The foregoing results in a minimum number of cycle overheads since all the Huffman codes would be decoded in a single lookup, itself. However, the foregoing results in a significant amount of memory wastage. The memory wastage becomes more apparent if very long Huffman codes with a maximum length of 19 are used, as in MPEG-2. Where Huffman codes with a maximum length of 19 are used, 500K (2^{19}) memory locations are used. As can be seen, the memory consumption increases exponentially. To reduce the memory consumption, each Huffman code can be decoded with a different number of lookups, depending upon the Huffman code length. So, large codewords take more cycles than shorter code words. However, the larger codewords occur less often than smaller codewords in a stream. Accordingly, the average number of cycles needed to decode all the Huffman codes is reasonably small, with respect to the total memory consumed to pack all the Huffman codes along with their data symbols and

signal bits for all the spectral amplitudes in a given frame.

[0025] In the exemplary illustrated Huffman code, there are a different number of code words with different lengths. For example, there is one codeword with two bits, four with three bits, two with four bits, four with five bits, and two with six bits. A binary tree can be generated that contains all the Huffman codes.

[0026] Referring now to **FIGURE 2**, there is illustrated an illustration of the binary tree for the Huffman code described in **FIGURE 1**. The binary tree includes a root node 405, intermediate nodes 410, and all the Huffman codes, with the data symbols A,B,...L, attached at the ends. However, the data symbols A,B,...L traverse a variable number of total branches and each level has a different number of data symbols. For example, the data symbol A traverses a total of two branches, whereas the data symbol K and L traverse a total of six branches. Additionally, level 2 has one symbol, A, level 3 has four symbols, B,C,D, and E, level 4 has two symbols, F, and G, level 5 has four symbols, H,I, and J, and level 6 has two symbols, K and L.

[0027] Various combinations of memory consumption and cycles overhead are possible by cutting the root tree at different levels. Repeating the same procedure by cutting sub-trees (trees emerging from intermediate nodes 410) and keeping link addresses from the parent tree, it is possible to cut the child trees at different levels and keep doing so until all the Huffman codes are covered with a reasonable memory consumption and require a different number of lookups per Huffman code. The cut levels in the tree can be appropriately selected to pack all the Huffman

codes and their corresponding data symbols in an efficient way, depending upon the memory requirements and cycles available.

[0028] To cut the levels in the tree, a proportion P is defined which is ratio of actual number of branches present up to that level to the total number of branches possible up to that level. To compute total branches possible up to any Nth level is $2*((2^N)-1)$. For example, at level 1 P(1) = 1 (2/2), because there are two actual branches and maximum two possible branches. At level 2, P(2) = 1 (6/6), because there are 6 actual branches and 6 $(2*((2^2)-1)$ maximum possible branches. For level 3, P(3) = 0.8571 (12/14) because there are total 12 branches up to that level and there are 14 ($2*(2^3-1)$) maximum branches possible up to that level. For level 4, P(4) = 0.5333 (16/30). For level 5, P(5) = 0.3225(20/62).

[0029] As can be seen, as the levels increase the proportion 'P' will decrease. A threshold T is selected, such that when the proportion P(N) falls below the threshold T, the tree is cut at level N. When a tree is cut at a level, N, each of the nodes at level N becomes the root of a new binary tree.

[0030] Additionally 2^N memory locations are associated with each of the possible N-bit combinations. Where a data path associated with an N-bit combination leads to a symbol, the memory location associated with the N-bit combination stores the symbol. Where the data path associated with an N-bit combination leads to a new binary tree, the memory location associated with the N-bit combination stores a link to the new tree. For any N-bit combination that exceeds a codeword for a symbol, where the codeword matches a prefix of the N-bit combination, the memory location

associated therewith indicates the foregoing and the symbol.

[0031] Referring now to **FIGURE 3**, there is illustrated a block diagram describing the binary tree of **FIGURE 2**, packed into a memory in accordance with an embodiment of the present invention. The threshold T = 0.58 is selected. As noted above, the P(3) = 0.8571, and P(4) = 0.5333. Accordingly, the tree is cut at level 4, resulting in new trees at root nodes 410(0)' and 410(1)'.

[0032] The value P is calculated for each level of the new trees at root nodes 410(0)' and 410(1)'. For the new tree at root node 410(0)', at level 1, P(1) = 1, and at level 2, there are no branches. For the new tree at root node 410(1)', at level 1, P(1) = 1 because there are two branches and two possible branches. At level 2, P(2) = 1 because there are six branches and six possible branches. At level 3, P(3) = 0.571. Accordingly, the tree at root node 410(1)' is cut at level 3, and a new tree emerges from a root at 410(0)".

[0033] The binary tree from root node 410, cut at level 3, is then packed into eight memory locations 505(000)...505(111). Each possible three bit combination is associated with a memory location 505(000)...505(111), where 505(bbb) is associated with the three bit combination bbb.

[0034] The data paths associated with the three-bit combinations, 010, 100, 101, and 110, result in symbols B, C, D, and E. Accordingly, memory locations 505(010), 505(100), 505(101), and 505(110) store the symbols B, C, D, and E, respectively. The data paths associated with the three-bit combinations, 011 and 111, lead to the new trees resulting at 410(0)', and 410(1)', respectively. Accordingly, memory locations 505(011) and 505(111) store a

link to the new binary trees resulting at 410(0)' and 410(1)', respectively.

[0035] However, it is possible that some memory locations, e.g., 505(000) and 505(001), will be associated with three bit combinations that are not associated with a particular path. This is because certain data paths do not exist. Certain data paths do not exist where a codeword matches the prefix of the data path. For example, there are no data paths for 000 and 001. This is because the codeword 00, corresponding to symbol A, matches the prefix for 000 and 001. Accordingly, the memory locations 505(000) and 505(001) store an indicator that the codeword 00 = A.

[0036] The tree at root node 410(0)' is packed into two memory locations, 510(0) and 510(1). Memory locations 505(0) and 505(1) store symbols C and D, respectively.

[0037] The tree at root node 410(1)' is packed into four memory locations, 515(00), 515(01), 515(10), and 515(11). The data paths associated with the two bit combinations 00, 01, and 10 result in symbols H, I, and J. Accordingly, memory locations 515(00), 515(01), and 515(10) store symbols H, I, and J, respectively. The data path associated with the two-bit combination 11, leads to the new binary tree at 410(0)". Accordingly, memory location 515(11) stores a link to table 520.

[0038] The tree at node 410(0)" is packed into two memory locations 520(0) and 520(1). Memory locations 520(0) and 520(1) store symbols K and L, respectively.

[0039] Referring now to **FIGURE 4**, there is illustrated a flow diagram for packing a binary tree. At 605, a value N is set equal to 1, and a threshold T is selected. At 610, a proportion P of the total number of branches at level N divided by the total number of possible nodes at level N,

$2*(2^N-1)$ is calculated for level N. If P > T at 620, then N = N+1 at 625, and 610-525 are repeated.

[0040] When the proportion P < T, the tree is cut at level N (630). When the tree is cut at level N, each node becomes the root of a new binary tree. For combinations of N-bits associated with data paths that lead to a symbol, a memory location is associated therewith, storing the symbol (635). For each N-bit combination associated with a data path that leads to a new binary tree, a memory location is associated therewith, storing a link to the new binary tree (640). For each N bit combination that is not associated with a data path, there is a codeword that matches the prefix of the N bits (645). Accordingly, the codeword and the symbol associated with the codeword that matches the prefix of the N bits are stored in each of the memory locations associated therewith. During 632, for each new binary tree that results from 630, 605-645 are recursively repeated.

[0041] Referring now to **FIGURE 5**, there is illustrated a block diagram describing decoding a Huffman variable length code from memory in accordance with an embodiment of the present invention. The Huffman code tables are packed as described in **FIGURE 4**.

[0042] At 705, a bit stream comprising variable length codes is read serially into a bit buffer register. At 710, a n_i (where n is a variable) bits X_i are read from the portion of the bit stream. The value of $n_i = \log_s$(length of binary tree in iteration i). For example, in the case where the Huffman codes are as shown in **FIGURE 3**, at the starting of decoding a new symbol, the first binary tree at node 410 has eight entries. Accordingly, n=3. At 715, the bits X_i are used to address memory packing the binary tree. At 720, the contents of the memory location referenced by the bits X_i

are read to determine whether a codeword match has been found.

[0043] A match is determined if the first M bits of bits X_i match that of an M-bit codeword. If a match occurs, the last N-M bits of bits X_i are returned (725) to the bit buffer register and the decoded symbol is output (730).

[0044] If not match is found, but a link to another a memory storing another binary is found, the bit X_i are released (735) and 710 is repeated. The foregoing is repeated until a symbol match is found. Alternatively, if the bits do not match with the bits of any codeword after Σn_i exceeds the number of bits in the maximum length code word. An error condition is declared and the bit register is cleared (740).

[0045] Referring now to **FIGURE 6**, a representative hardware environment for a computer system 58 for practicing the present invention is depicted. A CPU 60 is interconnected via system bus 62 to random access memory (RAM) 64, read only memory (ROM) 66, an input/output (I/O) adapter 68, a user interface adapter 72, a communications adapter 84, and a display adapter 86. The input/output (I/O) adapter 68 connects peripheral devices such as hard disc drives 40, floppy disc drives 41 for reading removable floppy discs 42, and optical disc drives 43 for reading removable optical disc 44 (such as a compact disc or a digital versatile disc) to the bus 62. The user interface adapter 72 connects devices such as a keyboard 74, a mouse 76 having a plurality of buttons 67, a speaker 78, a microphone 82, and/or other user interfaces devices such as a touch screen device (not shown) to the bus 62. The communications adapter 84 connects the computer system to a

data processing network 92. The display adapter 86 connects a monitor 88 to the bus 62.

[0046] An embodiment of the present invention can be implemented as a file resident in the random access memory 64 of one or more computer systems 58 configured generally as described in **FIGURE 6**. Until required by the computer system 58, the file may be stored in another computer readable memory, for example in a hard disc drive 40, or in removable memory such as an optical disc 44 for eventual use in an optical disc drive 43, or a floppy disc 42 for eventual use in a floppy disc drive 41. The file can contain a plurality of instructions executable by the computer system, causing the computer system to perform various tasks, such effectuating the flow chart described in **FIGURES 4** and **5**.

[0047] One embodiment of the present invention may be implemented as a board level product, as a single chip, application specific integrated circuit (ASIC), or with varying levels integrated on a single chip with other portions of the system as separate components. The degree of integration of the system will primarily be determined by speed and cost considerations. Because of the sophisticated nature of modern processors, it is possible to utilize a commercially available processor, which may be implemented external to an ASIC implementation of the present system. Alternatively, if the processor is available as an ASIC core or logic block, then the commercially available processor can be implemented as part of an ASIC device with various functions implemented as firmware.

[0048] Common compression standards such as a MPEG-2 utilize Huffman code for variable length coding. A decoder

system can comprise memory packing Huffman codes as described herein. Additionally, the decoder system can decode the Huffman codes as described in **FIGURE 5**.

[0049] Referring now to **FIGURE 7**, there is illustrated a block diagram of an exemplary decoder for decoding compressed video data, configured in accordance with an embodiment of the present invention. A processor, that may include a CPU, reads a stream of transport packets (a transport stream) into a transport stream buffer 203 within an SDRAM 201. The data is output from the transport stream presentation buffer 203 and is then passed to a data transport processor 205. The data transport processor then demultiplexes the MPEG transport stream into its PES constituents and passes the audio transport stream to an audio decoder 215 and the video transport stream to a video transport processor 207. The video transport processor 207 converts the video transport stream into a video elementary stream and provides the video elementary stream to an MPEG video decoder 209 that decodes the video.

[0050] The audio data is sent to the output blocks and the video is sent to a display engine 211. The display engine 211 is responsible for and operable to scale the video picture, render the graphics, and construct the complete display among other functions. Once the display is ready to be presented, it is passed to a video encoder 213 where it is converted to analog video using an internal digital to analog converter (DAC). The digital audio is converted to analog in the audio digital to analog converter (DAC) 217.

[0051] Referring now to **FIGURE 8**, there is illustrated a block diagram describing an audio decoder 215 or video decoder 209 in accordance with an embodiment of the present invention. The audio decoder or video decoder comprises a

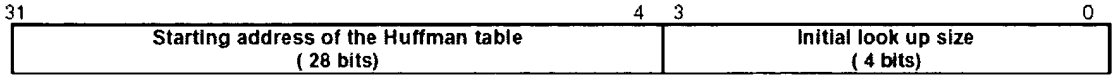
Huffman Decoder 285, an inverse quantizer 287, and an inverse frequency domain converter 290. The inverse frequency domain converter 290 converts frequency domain samples to either the spatial domain or the time domain.

[0052] In the case of a video decoder 209, the inverse frequency domain converter 290 comprises an inverse discrete cosine transformation block and converts frequency domain samples to the spatial domain. In the case of an audio decoder 215, the inverse frequency domain converter 290 comprises an inverse modified discrete cosine transformation block and converts frequency domain samples to the time domain.

[0053] The elementary stream is received by the Huffman Decoder 285. The Huffman Decoder 285 decodes Huffman coded symbols in the elementary stream. The inverse quantizer 290 dequantizes frequency domain coefficients in the elementary stream. The inverse frequency domain converter block 290 converts the frequency domain coefficients to either the spatial or time domain.

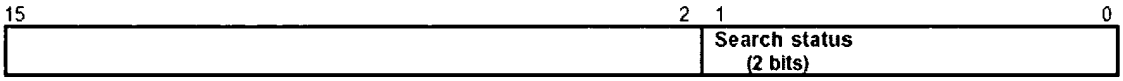
[0054] The Huffman decoder 285 can comprise memory that packs Huffman codes as described herein. Additionally, the Huffman decoder 285 can effectuate the flow chart described in **FIGURE 5** to decode Huffman codes. When the Huffman decoder 285 finds a symbol for a code word, the symbol can be used to calculate spectral values. If the Huffman table is an unsigned table, then the next bits in the stream are read from the bit stream as sign bits. Alternatively, if an escape symbol is found, a pulse data spectral coefficient is calculated and then the search is directed back until all of the spectral coefficients for the present frame are computed.

[0055] For decoding any Huffman table, a multi-stage lookup with a different number of bits looked up at each stage is done. This gives an advantageous tradeoff between the total memory requirement and the worst case cycle counts for decoding any Huffman code. Assuming a 32-bit register contains a value for the starting address of the table lookup and initial lookup size, the bit interpretation can be as show in the table.



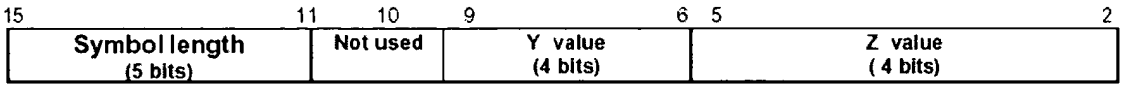
[0056] The magnitude of all data fields, signs for all the data fields (wherever applicable), Huffman code length, next stage lookup address or next stage size can be stored in a 16-bit word as described below.

HUFFMAN TABLES PACKING FOR MPEG-1 LAYER-3 AUDIO DECODER

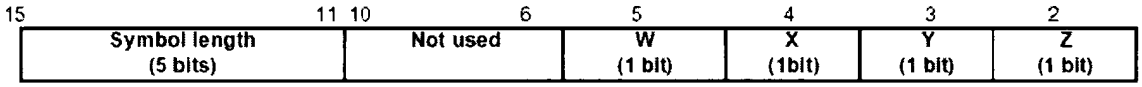


[0057] The interpretation of bits from 2 to 15 can be dependent on the value of bits 0 and 1. To decode a Huffman code, the four least significant bits in the 32-bit register indicate how many bits from the data stream are extracted for Huffman decoding. The contents of the memory location in the memory packing a binary tree that corresponds to the extracted bits can then be examined. If the value of bits 1 and 0 is 00, then there is an error in the Huffman stream and the decoder system takes appropriate actions.

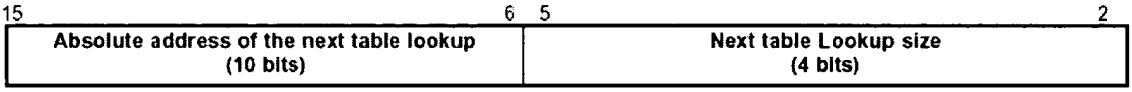
[0058] If the value of search status (bits 1 and 0) is 10, it is a valid Huffman code and interpretation of the bits from 2 to 15 are given below for each lookup table. On decoding the Huffman code, the length of the Huffman code is returned back and the pointer is set appropriately to extract the appropriate number of bits for decoding the next Huffman code.



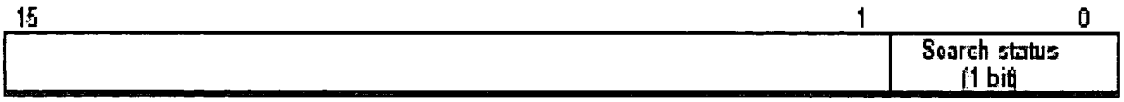
The interpretation for special tables 33 and tables 34 (Dimension 4) used in MPEG-1 Layer 3 decoder is as follows:



If the value of search status is 01, then it is an intermediate link and interpretation of the bits from 2 to 15 is as follows:



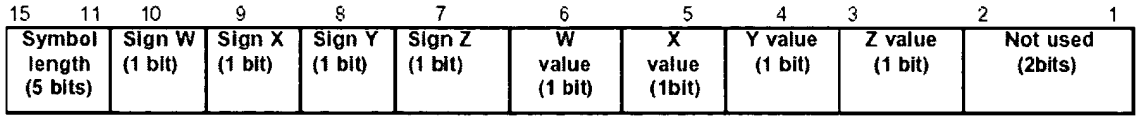
HUFFMAN TABLES PACKING FOR MPEG-2 AAC AUDIO DECODER



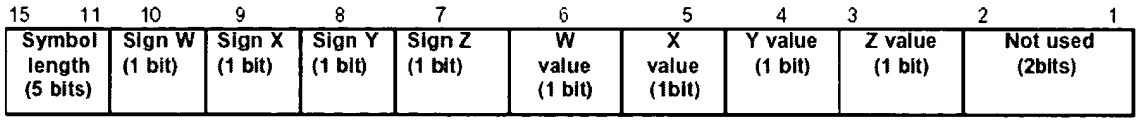
[0059] To decode a Huffman code, the four least significant bits in the 32-bit register indicate how many bits from the data stream are extracted for Huffman decoding. The contents of the memory location in the memory packing a binary tree that corresponds to the extracted bits can then be examined.

[0060] If the value of bit 0 is 0, then it is a code or error. To distinguish between an error or a code, the corresponding Huffman code length value is decoded. If the value of the code length is larger, that is not defined at all in the standard Huffman table, then an error is returned and a appropriate action is taken by the decoder. Otherwise, the interpretation of the bits from 1 to 15 are given below for each Huffman table. On decoding the Huffman code, the length of the Huffman code is returned back and pointer is set appropriately to extract the appropriate number of bits for decoding the next Huffman code.

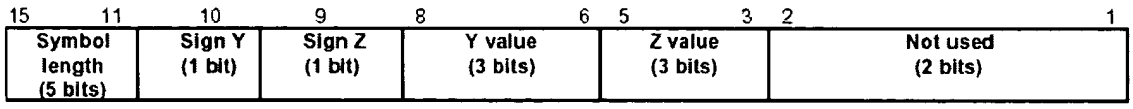
For MPEG-2-AAC Huffman table 1 and 2:



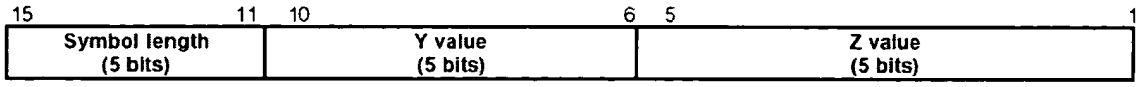
For MPEG-2-AAC Huffman table 3 and 4:



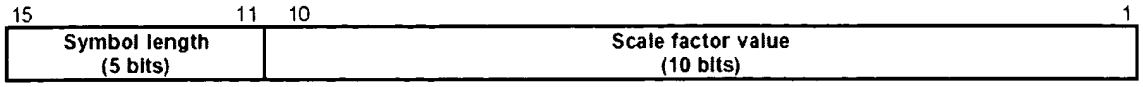
For MPEG-2-AAC Huffman table 5 and 6:



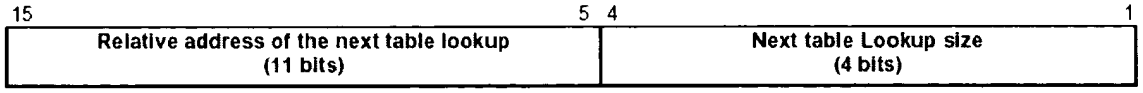
For MPEG-2-AAC Huffman table 7 through 11:



For Huffman Scale Factor Table:



If the value of the search status (i.e., bit 0) is 1, then it is an intermediate link and the interpretation of the bits from 1 to 15 is given below:



[0061] While the invention has been described with reference to certain embodiments, it will be understood by those skilled in the art that various changes may be made and equivalents may be substituted without departing from the scope of the invention. In addition, many modifications may be made to adapt particular situation or material to the teachings of the invention without departing from its scope. Therefore, it is intended that the invention not be limited to the particular embodiment(s) disclosed, but that the invention will include all embodiments falling within the scope of the appended claims.